A Small Minimal Aperiodic Reversible Turing Machine

Julien Cassaigne, Nicolas Ollinger, Rodrigo Torres

▶ **To cite this version:**

Julien Cassaigne, Nicolas Ollinger, Rodrigo Torres. A Small Minimal Aperiodic Reversible Turing Machine. Journal of Computer and System Sciences, 2017, 84, pp.288-301. 10.1016/j.jcss.2016.10.004 . hal-00975244

HAL Id: hal-00975244
https://hal.science/hal-00975244

Submitted on 8 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Small Minimal Aperiodic Reversible Turing Machine

Julien Cassaigne[a], Nicolas Ollinger[c,1], Rodrigo Torres-Avilés[b,1,2,*]

[a] *Institut de Mathématiques de Marseille (UMR 7373), Université de Marsseille, 13288 Marseille Cedex 9, France*
[b] *Departamento de Ingeniería Matemática and Centro de Investigación en Ingeniería Matemática (CI²MA), Universidad de Concepción,Centro de Modelamiento Matemático (CMM), Universidad de Chile, Casilla 160-C, Concepción, Chile*
[c] *Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 Orléans, France*

Abstract

A simple reversible Turing machine with four states, three symbols and no halting configuration is constructed that has no periodic orbit, simplifying a construction by Blondel, Cassaigne and Nichitiu and positively answering a conjecture by Kari and Ollinger. The constructed machine has other interesting properties: it is symmetric both for space and time and has a topologically minimal associated dynamical system whose column shift is associated to a substitution. Using a particular embedding technique of an arbitrary reversible Turing machine into the one presented, it is proven that the problem of determining if a given reversible Turing machine without halting state has a periodic orbit is undecidable.

Keywords: Turing machines, discrete dynamical systems, symbolic dynamics.

The present article is mainly devoted to show a particular Turing machine without halting configuration nicknamed "SMART" and to establish its nice dynamical properties:

- reversible,
- both space and time symmetric,
- without periodic points (aperiodic),
- topologically transitive and minimal,
- with a substitutive trace shift, and
- small: four states and three symbols,

Almost periodic points exists in any metric dynamical system [1], but several examples of systems without periodic points exist. Nevertheless, when in 1997 Kůrka considered Turing machines from a dynamical systems point of view, he conjectured that the existence of periodic points in this model was necessary. The intuition suggests that a non-trivial Turing machine will always need to cover arbitrary long distances to avoid temporal periodicity, and most machines do this by regularly moving over periodic configurations. A counterexample was found later by Blondel, Cassaigne and Nichitiu [2] and the approach taken by Kůrka gave birth to a series of publications about dynamical properties of Turing machines [3, 4, 5, 6].
Considering Turing machines as dynamical systems has several motivations (and difficulties). Motivations are related to the study of *complexity* of dynamical systems in the sense of predictability of the system

*Corresponding author
Email addresses: julien.cassaigne@univ-mrs.fr (Julien Cassaigne), nicolas.ollinger@univ-orleans.fr (Nicolas Ollinger), rtorres@ing-mat.udec.cl (Rodrigo Torres-Avilés)

[1]This work has been supported by ECOS Sud – CONICYT project C12E05
[2]This work has been supported by CONICYT BASAL project CMM, Universidad de Chile, and CI²MA, Universidad de Concepción.

behaviour from its description and initial conditions. If a universal Turing machine is embedded into a given dynamical system, this one inherits the complexity of the machine, and in particular several questions about the system result to be undecidable. But typical undecidable problems about Turing machines are not always natural for dynamical systems. One of the drawbacks comes from the existence, in the context of Turing machines, of a *finite* initial configuration and a starting and a halting state. Which, in terms of dynamical systems, imply to work over a particular subset of points (usually a *non-compact* subset), and to be limited to systems that *halt* at some points (in particular when starting at the halting state). Kůrka looked at Turing machines directly as dynamical systems by omitting the halting and starting states and eliminating the notion of blank state to work over arbitrary initial tapes.

In this new context, the difficulty comes from the fact that very few undecidable problems consider the behavior of machines without initial state and starting from potentially infinite configurations. Surprisingly enough, the first problem of this kind was studied and proved undecidable in 1966 by Hooper [7]. The *immortality* problem asks for the existence of initial configurations where the machine does not halt (or where a given state is never attained). We remark that it is different from the *totality* problem, that asks for the existence of a *finite* configuration where the machine does not halt (for example, a machine that goes to the left over any symbol but halts when attaining the blank symbol is total, but not mortal). Defining mortal machines requires to avoid *unbounded searches* through a repetitive movement which would lead to immortal trajectories. Hooper uses a technique of *recursive calls* which implements unbounded searches by performing imbricate calls to searches of length three. This is the technique used in [2] to define a *complete* (without halting states) Turing machine without periodic points. There, the undecidability of the problem of existence of periodic points for complete Turing machines is also proved.

Another achievement in this direction is the undecidability of the periodicity problem, established by Kari and Ollinger [8]. They focus on the restricted class of *reversible* machines. It is not necessary to recall the importance of "reversibility" in the context of computer science and dynamical systems, but we would like to point out its relation to other properties. A system is called "reversible" when the precedent configuration is uniquely defined (if it exists). In the case of complete Turing machines, it results to be equivalent to bijectivity and surjectivity of the global transition function, and it can be easily checked from the machine transition rule. Surjectivity is a necessary condition for *transitivity*, which says that, for every pair of points, there is a third point that passes as close as we want to both. A system which is transitive has a high uniformity, since most points pass near every other point and have, then, a similar behaviour over an arbitrary long time.

Minimality is a stronger property. A minimal system is a system without proper subsystems, *i.e.*, with no topologically closed subsets which are also closed for the dynamics. In a minimal system, every point passes, as close as we want, to any other point. Minimal systems have no periodic orbits, because these are proper subsystems. Thus aperiodic and reversible machines are good candidates to be minimal systems. The machines defined in [2] are not reversible, and the machines defined in [8] are not complete, thus no dynamical system can be defined from them. The methods used in these papers were not directly adaptable to the construction of such machines, and their existence was stated as an open question in [8].

The SMART machine answers this question positively and the associated system also results to be minimal. Usually minimal systems come from *substitutions* and SMART is not the exception. In Section 2 a subshift is associated to SMART, following [5], and it is proved that it is a *substitutive subshift* of a primitive substitution of non-constant length that we exhibit. Stronger than reversibility, time-symmetry says that the inverse of a function is conjugated with the function itself through an involution, which in this context means that the reverse of the Turing machine has the same rule that the direct machine, up to a transposition of the symbols and states. SMART is also time symmetric.

The existence of SMART allows the definition of a big family of aperiodic machines in Section 3, from which the problem of existence of periodic points, restricted to the context of reversible and complete Turing Machines, is proved to be undecidable.

1. A small aperiodic complete and reversible Turing machine

In this section we present the SMART machine and prove its aperiodicity. The proof mainly follows the scheme developed in [2].

1.1. Preliminary definitions and the machine

1.1.1. Word notation.

In this work, Σ^* denotes the set of finite sequences of elements of Σ, called *finite words*. Also, Σ^ω represents the set of right-infinite sequences of elements of Σ, called *semi-infinite words*, and ${}^\omega\Sigma$ denotes the left-infinite sequences of Σ. Finally, $\Sigma^\mathbb{Z}$ denotes the set of *bi-infinite words*.

1.1.2. Turing Machine.

A Turing machine (TM) M is a tuple $M = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite set of symbols, and $\delta \subseteq Q \times \Sigma \times \Sigma \times Q \times \{-1, 0, +1\}$ is the transition table. M is said to be *deterministic* if δ is functional in its two first components; it is said to be *complete* if δ is total in its two first components.

1.1.3. Configuration of a Turing machine.

A *configuration* is a tuple (w, i, r) of $\Sigma^\mathbb{Z} \times \mathbb{Z} \times Q$. A *finite* configuration is a tuple (v, i, r) of $\Sigma^m \times \{0, 1, ..., m-1\} \times Q$, for some $m \in \mathbb{N}$. A *right semi-finite* configuration is a tuple (u, i, r) of $\Sigma^\omega \times \mathbb{N} \times Q$. A *left semi-finite* configuration is an element (u, i, r) of ${}^\omega\Sigma \times -\mathbb{N} \times Q$.

1.1.4. Instructions of a Turing machine.

An instruction (r, s, s', r', c) can be applied to a configuration (w, i, r'') if $w_i = s$ and $r'' = r$, leading to the configuration $(w', i + c, r')$, where $w'_i = s'$ and $w'_k = w_k$ for all $k \neq i$. If the configuration is finite or (right or left) semi-finite with domain K, and $i + c \notin K$, then the instruction cannot be applied and the machine halts. If no instruction can be applied, the machine halts too.
It is said that the instruction (r, s, s', r', c) *starts* from the state r and *goes* to state r'.
We say that M *reaches configuration* y *from* x if y is the result of evolving M on x over a finite number of steps, and we denote this by $x \vdash^* y$. This notion is considered for finite, semi-finite and infinite configurations.
M is said to be *reversible* if each configuration has at most one immediate predecesor.
In the computational context, Turing machines have an initial and a final state, but since we are studying its dynamics, we omit these definitions.

1.1.5. The SMART machine.

The machine which is the object of this article is described in figure 1. We remark the symmetry between states b and d, and between p and q. State b writes the same as d on the tape, but it goes in the opposite direction of d. The analogous occurs with p and q.

1.2. Basic movements of SMART

The behaviour of SMART consists in recursively applying different types of *bounded searches*. These can be described by the next four propositions. They say that the machine finally transverse every block of 0s. But in the proof of these propositions, we can see that it does it by recursive calling smaller bounded searches in a nested way.

$B(n)$: $(\forall s_+ \in \{1,2\})(\forall s_* \in \{0,1,2\})$ $\begin{pmatrix} s_* & 0^n & 0 & s_+ \\ & & b & \end{pmatrix} \vdash^* \begin{pmatrix} s_* & 0^{n+1} & s_+ \\ & b & & \end{pmatrix}$

$D(n)$: $(\forall s_+ \in \{1,2\})(\forall s_* \in \{0,1,2\})$ $\begin{pmatrix} s_+ & 0 & 0^n & s_* \\ & d & & \end{pmatrix} \vdash^* \begin{pmatrix} s_+ & 0^{n+1} & s_* \\ & & d & \end{pmatrix}$

$P(n)$: $(\forall s_+ \in \{1,2\})$ $\begin{pmatrix} 0 & 0^n & s_+ \\ p & & \end{pmatrix} \vdash^* \begin{pmatrix} 0^{n+1} & s_+ \\ & p \end{pmatrix}$

$Q(n)$: $(\forall s_+ \in \{1,2\})$ $\begin{pmatrix} s_+ & 0^n & 0 \\ & & q \end{pmatrix} \vdash^* \begin{pmatrix} s_+ & 0^{n+1} \\ q & \end{pmatrix}$

Lemma 1. *$B(n)$, $D(n)$, $P(n)$ and $Q(n)$ are true for all $n \in \mathbb{N}$.*

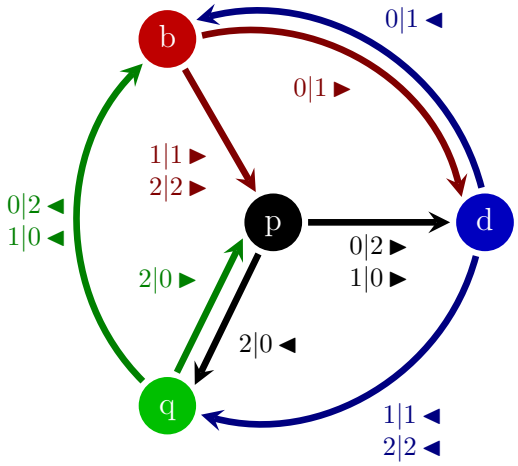

Figure 1: The SMART machine. An arrow from r to r' labelled $s|s'c$ represents the instruction (r, s, s', r', c) of the machine.

Proof. We make an induction over n. The basis can be done by hand by simulating the machine.
Let us take $n \geq 1$. We will do the proofs just for $B(n)$ and $P(n)$, because $D(n)$ and $Q(n)$ are symmetric.
Let us suppose that $B(n-1)$, $D(n-1)$, $P(n-1)$ and $Q(n-1)$ are true. First we prove $B(n)$.

$$
\begin{array}{l}
\left(\begin{array}{c} s_* \; 0 \; 0 \; 0^{n-2} \; 0 \; s_+ \\ \qquad\qquad\quad b \end{array} \right) \\
\qquad\qquad \text{Apply } B(n-1) \\
\left(\begin{array}{c} s_* \; 0 \; 0 \; 0^{n-2} \; 0 \; s_+ \\ \quad b \end{array} \right) \\
\qquad\qquad \text{One step} \\
\left(\begin{array}{c} s_* \; 1 \; 0 \; 0^{n-2} \; 0 \; s_+ \\ \quad\;\; d \end{array} \right) \\
\qquad\qquad \text{Apply } D(n-1) \\
\left(\begin{array}{c} s_* \; 1 \; 0 \; 0^{n-2} \; 0 \; s_+ \\ \qquad\qquad\quad d \end{array} \right) \\
\qquad\qquad \text{One step} \\
\left(\begin{array}{c} s_* \; 1 \; 0 \; 0^{n-2} \; 0 \; s_+ \\ \qquad\qquad\quad q \end{array} \right) \\
\qquad\qquad \text{Apply } Q(n-1) \\
\left(\begin{array}{c} s_* \; 1 \; 0 \; 0^{n-2} \; 0 \; s_+ \\ \quad q \end{array} \right) \\
\qquad\qquad \text{One step} \\
\left(\begin{array}{c} s_* \; 0 \; 0 \; 0^{n-2} \; 0 \; s_+ \\ \quad b \end{array} \right)
\end{array}
\tag{1}
$$

Now, for $P(n)$:

$$
\begin{array}{l}
\left(\begin{smallmatrix} 0\ 0\ 0^{n-2}\ 0\ s_+ \\ p \end{smallmatrix} \right) \\
\qquad \text{One step} \\
\left(\begin{smallmatrix} 2\ 0\ 0^{n-2}\ 0\ s_+ \\ d \end{smallmatrix} \right) \\
\qquad \text{Apply } D(n-1) \\
\left(\begin{smallmatrix} 2\ 0\ 0^{n-2}\ 0\ s_+ \\ d \end{smallmatrix} \right) \\
\qquad \text{One step} \\
\left(\begin{smallmatrix} 2\ 0\ 0^{n-2}\ 0\ s_+ \\ q \end{smallmatrix} \right) \\
\qquad \text{Apply } Q(n-1) \\
\left(\begin{smallmatrix} 2\ 0\ 0^{n-2}\ 0\ s_+ \\ q \end{smallmatrix} \right) \\
\qquad \text{One step} \\
\left(\begin{smallmatrix} 0\ 0\ 0^{n-2}\ 0\ s_+ \\ p \end{smallmatrix} \right) \\
\qquad \text{Apply } P(n-1) \\
\left(\begin{smallmatrix} 0\ 0\ 0^{n-2}\ 0\ s_+ \\ p \end{smallmatrix} \right)
\end{array}
\tag{2}
$$

□

It is important to remark that applying $B(n)$ always implies to apply $B(n-1)$ as the first step, and also $B(n-2)$ and $B(0)$. Analogously, applying $P(n)$ implies to apply $P(n-1)$ and $P(0)$ as the last step. Interestingly, just before applying $P(0)$ by the last time, i.e., 3 steps before finishing $P(n)$, the head is on the rightmost 0 with state p. This will be used in the future. Thus, we define the next two propositions which also hold.

$P'(n)$: $(\forall s_+ \in \{1,2\})$ $\left(\begin{smallmatrix} 0\ 0^n\ s_+ \\ p \end{smallmatrix} \right) \vdash^* \left(\begin{smallmatrix} 0^n\ 0\ s_+ \\ p \end{smallmatrix} \right)$

$Q'(n)$: $(\forall s_+ \in \{1,2\})$ $\left(\begin{smallmatrix} s_+\ 0^n\ 0 \\ q \end{smallmatrix} \right) \vdash^* \left(\begin{smallmatrix} s_+\ 0\ 0^n \\ q \end{smallmatrix} \right)$

1.3. Aperiodicity

A *dynamical system* is a pair (X, T), where X is a set called phase space and $T : X \to X$ is called *global transition function*. A point x is called *periodic* for T if there exists $n \in \mathbb{N}$ such that $T^n(x) = x$.

We can associate a dynamical system to a Turing machine by describing the configuration set by $X = {}^{\omega}\Sigma \times Q \times \Sigma^{\omega}$, where the first component is the configuration to the left of the head, and the third component is the configuration to the right of the head. T consists in one application of δ over X, by shifting the tape and leaving the head at the center. This dynamical system is known as *Turing machine with moving tape (TMT)* and it was introduced in [9]. T is a total function only if M is complete. Reversibility of M results to be equivalent to biyectivity of T.

Following [8], we say that a Turing machine M is *aperiodic* if its associated moving tape system T has no periodic point. Let us remark that periodic points in the *moving tape* system can correspond to periodic configurations where the machine makes repetitive movements in a fixed direction. First, we prove the aperiodicity of two particular but important points.

Lemma 2. $\left(\begin{smallmatrix} s_+\ 0\ 0^n\ 1\ 0 \\ p \end{smallmatrix} \right) \vdash^* \left(\begin{smallmatrix} s_+\ 0\ 0^{n+1}\ 1 \\ p \end{smallmatrix} \right)$

Proof.

$$
\begin{array}{l}
\left(\begin{smallmatrix} s_+ & 0 & 0^n & 1 & 0 \\ & p & & & \end{smallmatrix} \right) \\
\qquad\qquad \text{Apply } P(n) \\
\left(\begin{smallmatrix} s_+ & 0 & 0^n & 1 & 0 \\ & & & p & \end{smallmatrix} \right) \\
\qquad\qquad \text{Two steps} \\
\left(\begin{smallmatrix} s_+ & 0 & 0^n & 0 & 1 \\ & & & b & \end{smallmatrix} \right) \\
\qquad\qquad \text{Apply } B(n+1) \\
\left(\begin{smallmatrix} s_+ & 0 & 0^n & 0 & 1 \\ b & & & & \end{smallmatrix} \right) \\
\qquad\qquad \text{One step} \\
\left(\begin{smallmatrix} s_+ & 0 & 0^n & 0 & 1 \\ & p & & & \end{smallmatrix} \right)
\end{array}
\tag{3}
$$

□

Lemma 3. *The semi-infinite configurations* $\left(\begin{smallmatrix} 0 & 0^w \\ b & \end{smallmatrix} \right)$ *and* $\left(\begin{smallmatrix} 0 & 0^w \\ p & \end{smallmatrix} \right)$ *are not periodic.*

Proof. Starting with any of these configurations, the machine makes a few steps (≤ 9), leading to $\left(\begin{smallmatrix} s_+ & 0 & 1 & 0^w \\ & p & & \end{smallmatrix} \right)$, with $s_+ \in \{1, 2\}$ depending on the initial state (if p then $s_+ = 2$, if b then $s_+ = 1$). Now we can apply Lemma 2 to see that the machine's evolution cannot be periodic. □

In order to generalize aperiodicity to any configuration, we will prove that, in the evolution of every configuration, arbitrary large blocks of 0s appear in a recurrent way.

Lemma 4. *If we define, for every* $n \geq 0$*, the set* $C_n = \{x \mid x \in \left(\begin{smallmatrix} s_+ & 0 & 0^n \\ q & & \end{smallmatrix} \right) \cup \left(\begin{smallmatrix} 0^n & 0 & s_+ \\ & p & \end{smallmatrix} \right)\}$*, then for every* $x \in C_n$*, either* x *or the orbit of* x *will eventually visit* C_m *for arbitrary large* m*.*

Proof. We just make the proof for initial state q, since p is symmetrical. Let us start with n maximal such that $x \in C_n$. If there is no maximal n, then x is already on every C_m.

$$
\begin{array}{l}
\left(\begin{smallmatrix} s_* & s_+ & 0 & 0^n & s_{++} \\ & & q & & \end{smallmatrix} \right) \\
\qquad\qquad \text{one step} \\
\left(\begin{smallmatrix} s_* & s_+ & 2 & 0^n & s_{++} \\ & b & & & \end{smallmatrix} \right) \\
\qquad\qquad \text{one step}
\end{array}
$$

If $s_+ = 1$		If $s_+ = 2$	
$\left(\begin{smallmatrix} s_* & 1 & 2 & 0^n & s_{++} \\ & & p & & \end{smallmatrix} \right)$		$\left(\begin{smallmatrix} s_* & 2 & 2 & 0^n & s_{++} \\ & & p & & \end{smallmatrix} \right)$	
	one step		one step
$\left(\begin{smallmatrix} s_* & 1 & 0 & 0^n & s_{++} \\ & & q & & \end{smallmatrix} \right)$		$\left(\begin{smallmatrix} s_* & 2 & 0 & 0^n & s_{++} \\ & & q & & \end{smallmatrix} \right)$	
	one step		one step
$\left(\begin{smallmatrix} s_* & 0 & 0 & 0^n & s_{++} \\ b & & & & \end{smallmatrix} \right)$		$\left(\begin{smallmatrix} s_* & 0 & 0 & 0^n & s_{++} \\ & & p & & \end{smallmatrix} \right)$	
	If $s_* \neq 0$		$P'(n)$
$\left(\begin{smallmatrix} s_* & 0 & 0 & 0^n & s_{++} \\ & p & & & \end{smallmatrix} \right)$		$\left(\begin{smallmatrix} s_* & 0 & 0^n & 0 & s_{++} \\ & & & p & \end{smallmatrix} \right)$	
	$P'(n+1)$		we are done
$\left(\begin{smallmatrix} s_* & 0 & 0^n & 0 & s_{++} \\ & & p & & \end{smallmatrix} \right)$			
	we are done		

Now we study the case $s_* = 0$.

$$\left(\begin{smallmatrix} 0\,0\,0\,0^n\ s_{++} \\ b \end{smallmatrix}\right)$$

one step

$$\left(\begin{smallmatrix} 1\,0\,0\,0^n\ s_{++} \\ d \end{smallmatrix}\right)$$

$D(n+1)$

$$\left(\begin{smallmatrix} 1\,0\,0\,0^n\ s_{++} \\ d \end{smallmatrix}\right)$$

one step

$$\left(\begin{smallmatrix} 1\,0\,0^n\ 0\ s_{++} \\ q \end{smallmatrix}\right)$$

$Q'(n+1)$

$$\left(\begin{smallmatrix} 1\,0\,0^n\ 0\ s_{++} \\ q \end{smallmatrix}\right)$$

we are done

□

Theorem 1. *The SMART machine has no periodic points.*

Proof. Consider an arbitrary configuration. After less than 9 steps, the head will be reading a 0 symbol in either state q or p, after which, by propositions P' and Q', it arrives to one of the sets C_n described in Lemma 4. The amount of 0s will grow then, expanding to the right or to the left. At some point, the machine will either reach a configuration of the form $\left(\begin{smallmatrix} 0\ 0^w \\ r \end{smallmatrix}\right)$, with $r \in \{b, p\}$ (or its symmetric), which we know to be aperiodic from Lemma 2, or it will pass by an infinite sequence of configurations of the form $\left(\begin{smallmatrix} 0\ 0^{n-1}\ s_+ \\ r \end{smallmatrix}\right)$ with $r \in \{b, p\}$ (or its symmetric), implying that its behaviour is not periodic. □

Corollary 1. *There exists a complete reversible Turing machine without any periodic point.*

2. Other properties of the SMART machine

2.1. More definitions

2.1.1. Subshifts, languages and subword.

The *shift* function σ, is defined in Σ^ω by $\sigma(w)_i = w_{i+1}$. Finite and semi-infinite words can be concatenated in a natural way. A finite word v is said to be a *subword* of another (finite or semi-infinite) word u, if there exist two indices i and j, such that $v = u_i u_{i+1} ... u_j$. In this case we write: $v \sqsubseteq u$. Given a subset $S \subseteq \Sigma^\omega$, a formal language is defined:

$$\mathcal{L}(S) = \{v \in \Sigma^* \mid (\exists u \in S)\ v \sqsubseteq u\}\ . \tag{4}$$

Reciprocally, given a formal language L, a set of infinite sequences can be defined:

$$\mathcal{S}_L = \{u \in \Sigma^\omega \mid (\forall v \sqsubseteq u)\ v \in L\}\ . \tag{5}$$

When S satisfies $\mathcal{S}_{\mathcal{L}(S)} = S$, it is called a *subshift*.

2.1.2. Metric

A metric is defined in Σ^ω by $d(u, v) = 2^{-n}$ if n is the smallest natural such that $u_n \neq v_n$. This metrics makes Σ^ω a compact space which means that every sequence has an accumulation point. Subshifts coincide with subsets of Σ^ω which are topologically closed and σ-invariant. Analogously, a metric can be defined in the phase space of a Turing machine with moving tape X: $d((u, r, u'), (v, r', v')) = max\{d(u, v), d(u', v')\} + \Delta_{rr'}$, where $\Delta_{rr'} = 1$ if $r \neq r'$ and 0 if $r = r'$. With this metric, X is compact and T is a continuous function, thus (X, T) is a *topological dynamical system*. The *cylinder* of a finite word v is $[v] = \{w \in \Sigma^\omega : w_0 \dots w_{|v|-1} = v\}$. It corresponds to all the elements in S which are at a distance less than or equal to $2^{-|v|}$ from any point that starts with v. Analogously, a finite configuration (v, r, v') will have an associated cylinder $[(v, r, v')] = \{(w, r, w') \in X : w_{|v|} \dots w_1 = v \wedge w'_0 \dots w'_{|v'|-1} = v'\}$.

2.1.3. The t-shift

Taking into account the notions previously introduced, we can define the projection $\pi : X \to Q \times \Sigma$ by $\pi(w', r, w'') = (r, w''_0)$. Now we define the sequence $\tau(x) = (\pi(T^n(x)))_{n\in\mathbb{N}}$, it is called the *trace* of x. The *t-shift* associated to T, denoted by $S_T \subseteq (Q \times \Sigma)^\omega$, is the image of τ: $S_T = \{\tau(x) \mid x \in X\}$. Since τ is continuous and surjective from (X, T) to $(S_T \subseteq (Q \times \Sigma)^\omega, \sigma)$, S_T is a subshift. Function τ can also be applied to finite and semi-finite configurations, with the consideration that, if the evolution stops, the obtained sequence will be finite.

2.1.4. Dynamical Properties

In a dynamical system (X, T), the orbit of a point is $O_T(x) = \{T(x), T^2(x), ...\}$.

A point x in a dynamical system (X, T) is *transitive* if for every cylinder U, there exists some natural n such that $T^n(x) \in U$. This is equivalent to saying that the orbit of x is topologically dense. If such a point exists, the system is said to be transitive.

A dynamical system (X, T) is said to be *minimal* if each of its points is transitive. It is equivalent to having no proper subsystem, *i.e.*, no proper subset $Y \subset X$ is topologically closed and T invariant.

A *substitution* ϕ is a morphism $\phi : A^* \to A^*$. It can be extended to A^ω. A semi-infinite word $w \in A^\omega$ is a *fixed point* of ϕ if $\phi(w) = w$. A *substitutive subshift* is the closure of the orbit of a fixed point of some substitution.

2.1.5. Time symmetry

The inverse of a reversible Turing machine is not exactly a Turing machine, since for the inverse machine rules are applied by moving before reading/writing. With this in mind, we remark that the rules of the inverse of a machine (Q, Σ, δ) (that we will call δ^{-1}) are obtained by "reversing" the original rules as follows.

$$(r', s', s, r, -c) \in \delta^{-1} \iff (r, s, s', r', c) \in \delta$$

A Turing machine is said to be *time-symmetric* if there exist involutions $h_Q : Q \to Q$ and $h_\Sigma : \Sigma \to \Sigma$ such that:

$$(h_Q(r), h_\Sigma(s), h_\Sigma(s'), h_Q(r'), c) \in \delta^{-1} \iff (r, s, s', r', c) \in \delta$$

Also, if we define $h : X \to X$ such that $h(w', r, w'') = (h_\Sigma(\dots w'_{-2}w'_{-1}), h_Q(r), h_\Sigma(w'_0 w''))$, if state r is reached from the left, or $h(w', r, w'') = (h_\Sigma(w'w''_0), h_Q(r), h_\Sigma(w''_1 w''_2 \dots))$, if state r is reached from the right; we have:

$$T(h(x)) = h(T^{-1}(x))$$

This is an adaptation of the definition of time symmetry in cellular automata, as seen in [10]. Let us remark that since h is *local*, it can also be applied to finite configurations.

2.2. SMART dynamical properties

Proposition 1. *The SMART machine is time-symmetric.*

Proof. Using involutions: $h_\Sigma(0) = 0$, $h_\Sigma(1) = 2$, $h_Q(d) = q$ and $h_Q(b) = p$, we will have that the SMART machine is time-symmetric as can be seen on the diagram of the inverse machine in Fig. 2. □

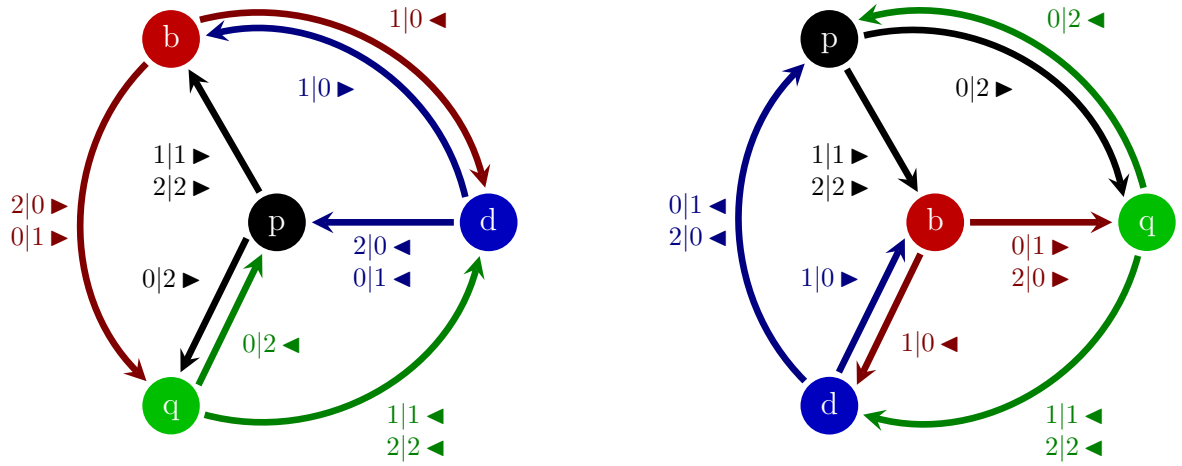

Figure 2: Two representations of the inverse of the SMART machine

With the previous result, we can now prove that every finite configuration can be reached from a block of 0s of the apropiate size.

Lemma 5. *For every finite word $v' \in \{0,1,2\}^*$ of length n, and every $i \in \{1,\dots,n\}$, there exist $k_1, k_2 \in \mathbb{N}$ and $r \in Q$ such that $\left(\begin{smallmatrix} 2\ 0^{n'} \\ b \end{smallmatrix}\ \begin{smallmatrix} 0\ 2 \end{smallmatrix}\right) \vdash^* \left(\begin{smallmatrix} 2^{k_1}\ v'_1 \cdots v'_i \cdots v'_n 2^{k_2} \\ r \end{smallmatrix}\right)$, where $n' = k_1 + k_2 + n - 3$.*

Proof. First, we will use the fact that the SMART machine is time-symmetric, so applying $h \circ T^t \circ h$ is the same as applying T^{-t}, for any time $t \in \mathbb{N}$. So now we just have to prove that $h\left(\begin{smallmatrix} 2^{k_1}\ v'_1 \cdots v'_i \cdots v'_n 2^{k_2} \\ r \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1^{k_1}\ h_\Sigma(v'_1) \cdots \cdots \cdots h_\Sigma(v'_n)\ 1^{k_2} \\ h_Q(r) \end{smallmatrix}\right)$ will eventually reach $h\left(\begin{smallmatrix} 2\ 0^{n'} \\ b \end{smallmatrix}\ \begin{smallmatrix} 0\ 2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1\ 0^{n'} \\ p \end{smallmatrix}\ \begin{smallmatrix} 0\ 1 \end{smallmatrix}\right)$.
Based on the proof of Theorem 1, we know that we can generate an increasing amount of 0 symbols in the tape. For this reason, we know that $\left(\begin{smallmatrix} 1\ h_\Sigma(v'_1) \cdots \cdots \cdots h_\Sigma(v'_n)\ 1 \\ h_Q(r) \end{smallmatrix}\right)$ will eventually reach one of the configurations considered in Lemma 4, with a block of 0s that will grow until arriving to one of the next configurations:

$$\left(\begin{smallmatrix} 1\ 0^j \\ p \end{smallmatrix}\ \begin{smallmatrix} 0\ v_1 \dots v_l\ 1 \end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix} 1\ 0 \\ q \end{smallmatrix}\ \begin{smallmatrix} 0^j\ v_1 \dots v_l\ 1 \end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix} 1\ v_1 \dots\ v_l \\ q \end{smallmatrix}\ \begin{smallmatrix} 0\ 0^j\ 1 \end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix} 1\ v_1 \dots v_l\ 0^j \\ p \end{smallmatrix}\ \begin{smallmatrix} 0\ 1 \end{smallmatrix}\right).$$

Thus, applying either $Q'(0)$ or $P'(0)$ we arrive to one of the next situations:

$$\underset{(i)}{\left(\begin{smallmatrix} 1\ 0^j \\ p \end{smallmatrix}\ \begin{smallmatrix} 0\ v_1 \dots v_l\ 1 \end{smallmatrix}\right)} \text{ or } \underset{(ii)}{\left(\begin{smallmatrix} 1\ 0 \\ q \end{smallmatrix}\ \begin{smallmatrix} 0^j\ v_1 \dots v_l\ 1 \end{smallmatrix}\right)} \text{ or } \underset{(iii)}{\left(\begin{smallmatrix} 1\ v_1 \dots\ v_l \\ q \end{smallmatrix}\ \begin{smallmatrix} 0\ 0^j\ 1 \end{smallmatrix}\right)} \text{ or } \underset{(iv)}{\left(\begin{smallmatrix} 1\ v_1 \dots v_l\ 0^j \\ p \end{smallmatrix}\ \begin{smallmatrix} 0\ 1 \end{smallmatrix}\right)}, \tag{6}$$

for some $v \in \{0,1,2\}^{n-j-1}$, and $v_1 \neq 0$ in situations (i) and (ii), and $v_l \neq 0$ in the other two.
In order to reach $\left(\begin{smallmatrix} 1\ 0^{n'} \\ p \end{smallmatrix}\ \begin{smallmatrix} 0\ 1 \end{smallmatrix}\right)$, we will need a certain amount of 1 symbols at the left or right of the initial configuration; this amount depends on v. Let $k(v)$ be the function that gives the amount of non-0 symbols in v. As before, we will do the proof only for configurations of the form (i) and (ii).

Case (i)

$$\left(\begin{smallmatrix} 1^{k(v)} & 1 & 0 & 0^j & v_1 & v_2 & \dots & v_l & 1 \\ & & & & p & & & & \end{smallmatrix}\right)$$

Regardless of the value of $v_1 \neq 0$

in 1 or 3 steps we reach the next, with $i = 0$ or 1

$$\left(\begin{smallmatrix} 1^{k(v)} & 1 & 0^{j-i+1} & 0 & 0^i & v_2 & \dots & v_l & 1 \\ & & & q & & & & & \end{smallmatrix}\right)$$

Apply $Q(j-i)$

$$\left(\begin{smallmatrix} 1^{k(v)} & 1 & 0^j & 0 & 0 & v_2 & \dots & v_l & 1 \\ & q & & & & & & & \end{smallmatrix}\right)$$

Two steps

$$\left(\begin{smallmatrix} 1^{k(v)} & 0 & 0^j & 0 & 0 & v_2 & \dots & v_l & 1 \\ & p & & & & & & & \end{smallmatrix}\right) \tag{7}$$

$P(j+2)$(or more if $v_2 = 0$)

$$\left(\begin{smallmatrix} 1^{k(v)} & 0 & 0^j & 0 & 0 & v_2 & .. & v_l & 1 \\ & & & & & p & & & \end{smallmatrix}\right)$$

applying the previous steps

iteratively $k(v) - 1$ times

$$\left(\begin{smallmatrix} 1 & 0 & 0^{n'} & 1 \\ & p & & \end{smallmatrix}\right)$$

Apply $P(n')$

$$\left(\begin{smallmatrix} 1 & 0 & 0^{n'} & 1 \\ & & & p \end{smallmatrix}\right)$$

Case (ii)

$$\left(\begin{smallmatrix} 1^{k(v)} & 1 & 0 & 0^j & v_1 & v_2 & \dots & v_l & 1 \\ & q & & & & & & & \end{smallmatrix}\right)$$

Two steps

$$\left(\begin{smallmatrix} 1^{k(v)} & 0 & 0 & 0^j & v_1 & v_2 & \dots & v_l & 1 \\ & p & & & & & & & \end{smallmatrix}\right) \tag{8}$$

$P(j+1)$

$$\left(\begin{smallmatrix} 1^{k(v)} & 0 & 0 & 0^j & v_1 & v_2 & \dots & v_l & 1 \\ & & & & p & & & & \end{smallmatrix}\right)$$

Which reduces to case 1

In this way, we have proved that, for (i) and (ii), we will always reach $\left(\begin{smallmatrix} 1 & 0^{n'} & 0 & 1 \\ & & p & \end{smallmatrix}\right)$, with $n' = j + 1 + k(v)$. For cases (iii) and (iv), we can assure, by symmetry, that the machine will reach $\left(\begin{smallmatrix} 1 & 0^{n'} & 0 & 1 \\ & q & & \end{smallmatrix}\right)$, then:

$$\left(\begin{smallmatrix} 1 & 1 & 0^{n'} & 0 & 1 \\ & q & & & \end{smallmatrix}\right)$$

Two steps

$$\left(\begin{smallmatrix} 1 & 0 & 0^{n'} & 0 & 1 \\ & p & & & \end{smallmatrix}\right) \tag{9}$$

Apply $Q'(n'+1)$

$$\left(\begin{smallmatrix} 1 & 0 & 0^{n'} & 0 & 1 \\ & & & p & \end{smallmatrix}\right)$$

Concluding that, for the last two cases, we need just one additional 1 symbol to reach the desired configuration. □

Lemma 6. *The orbits of configurations* $\left(\begin{smallmatrix} 0 & 0^\omega \\ r_1 & \end{smallmatrix}\right)$ *and* $\left(\begin{smallmatrix} {}^\omega 0 & 0 \\ & r_2 \end{smallmatrix}\right)$ *are dense in* X*, with* $r_1 \in \{b, p\}$ *and* $r_2 \in \{d, q\}$.

Proof. We just need to prove that any finite configuration can be reached from $\left(\begin{smallmatrix} 0 & 0^\omega \\ b & \end{smallmatrix}\right)$. The other cases can be proved by symmetry and time symmetry. Since any finite configuration can be reached from $\left(\begin{smallmatrix} 2\ 0^n & 0\ 2 \\ & b \end{smallmatrix}\right)$ for some n, we just need to prove that $\left(\begin{smallmatrix} 0 & 0^\omega \\ b & \end{smallmatrix}\right)$ can reach $\left(\begin{smallmatrix} 2\ 0^n & 0\ 2 \\ & b \end{smallmatrix}\right)$, for any $n \in \mathbb{N}$.

$$\left(\begin{smallmatrix} 0 & 0^\omega \\ b & \end{smallmatrix}\right)$$

One step

$$\left(\begin{smallmatrix} 1\ 0 & 0^\omega \\ d & \end{smallmatrix}\right)$$

Apply $D(n+3)$

$$\left(\begin{smallmatrix} 1\ 0\ 0^{n+1} & 0\ 0\ 0^\omega \\ & d \end{smallmatrix}\right)$$

One step

$$\left(\begin{smallmatrix} 1\ 0\ 0^{n+1} & 0\ 1\ 0^\omega \\ & b \end{smallmatrix}\right)$$

Apply $B(n+2)$

$$\left(\begin{smallmatrix} 1 & 0\ 0\ 0^{n+1}\ 1\ 0^\omega \\ b & \end{smallmatrix}\right)$$

Two steps

$$\left(\begin{smallmatrix} 1\ 2\ 0 & 0^{n+1}\ 1\ 0^\omega \\ d & \end{smallmatrix}\right)$$

Apply $D(n+1)$

$$\left(\begin{smallmatrix} 1\ 2\ 0^{n+1} & 0\ 1\ 0^\omega \\ & d \end{smallmatrix}\right)$$

Two steps

$$\left(\begin{smallmatrix} 1\ 2\ 0^n & 0\ 2\ 1\ 0^\omega \\ & b \end{smallmatrix}\right)$$

□

Corollary 2. *The SMART machine is transitive as well as its t-shift.*

Theorem 2. *The SMART machine is minimal as well as its t-shift.*

Proof. As we know from Lemma 4 and Theorem 1, we can "create" an arbitrary amount of 0s starting from any initial configuration. Now, from Lemma 5, if the correct amount of 0 is provided, we can reach any finite configuration. This proves that every point is transitive, and so SMART is minimal. Since its t-shift is a factor of (X, T), it inherits minimality. □

2.3. The t-shift is substitutive

In this part, we will prove that the t-shift is not only minimal, but also substitutive. For this, we will present a substitution and, with a pair of results, prove that the t-shift of SMART is the closure of the shift orbit of a fixed point of that substitution.
First, we recursively define the following functions.

- $B : \{1, 2\} \times \mathbb{N} \to (Q \times \Sigma)^*$
  $B(s_+, n) = B(s_+, n-1) \left(\begin{smallmatrix} 0 \\ b \end{smallmatrix}\right) D(1, n-1) \left(\begin{smallmatrix} s_+ \\ d \end{smallmatrix}\right) Q(1, n-1) \left(\begin{smallmatrix} 1 \\ q \end{smallmatrix}\right)$
  $B(s_+, 0) = \left(\begin{smallmatrix} 0 & s_+ & 1 \\ b & d & q \end{smallmatrix}\right)$
- $D : \{1, 2\} \times \mathbb{N} \to (Q \times \Sigma)^*$
  $D(s_+, n) = D(s_+, n-1) \left(\begin{smallmatrix} 0 \\ d \end{smallmatrix}\right) B(1, n-1) \left(\begin{smallmatrix} s_+ \\ b \end{smallmatrix}\right) P(1, n-1) \left(\begin{smallmatrix} 1 \\ p \end{smallmatrix}\right)$
  $D(s_+, 0) = \left(\begin{smallmatrix} 0 & s_+ & 1 \\ d & b & p \end{smallmatrix}\right)$
- $P : \{1, 2\} \times \mathbb{N} \to (Q \times \Sigma)^*$
  $P(s_+, n) = \left(\begin{smallmatrix} 0 \\ p \end{smallmatrix}\right) D(2, n-1) \left(\begin{smallmatrix} s_+ \\ d \end{smallmatrix}\right) Q(2, n-1) \left(\begin{smallmatrix} 2 \\ q \end{smallmatrix}\right) P(s_+, n-1)$
  $P(s_+, 0) = \left(\begin{smallmatrix} 0 & s_+ & 2 \\ p & d & q \end{smallmatrix}\right)$

- $Q : \{1,2\} \times \mathbb{N} \to (Q \times \Sigma)^*$
  $Q(s_+, n) = {}^{0}_{q} B(2, n-1) {}^{s_+}_{b} P(2, n-1) {}^{2}_{p} Q(s_+, n-1)$
  $Q(s_+, 0) = {}^{0\ s_+\ 2}_{q\ b\ p}$

Lemma 7. *$B(s_+, n)$ is the trace corresponding to applying proposition $B(n)$ to $\left({}^{s_*\ 0^n\ 0\ s_+}_{\quad\quad b\quad\quad} \right)$ until $\left({}^{s_*\ 0^{ns+1}\ s_+}_{\quad\quad b\quad\quad} \right)$. The analogous goes for $D(s_+, n)$, $P(s_+, n)$ and $Q(s_+, n)$.*

Proof. It is enough to see the proof of lemma 1 and take the trace. □

Now, let us define the substitution.

$\phi : (Q \times \Sigma)^* \to (Q \times \Sigma)^*$

$\phi({}^{0}_{b}) = {}^{0\ 0\ 1\ 1}_{b\ d\ b\ p} = {}^{0}_{b} D(1,0)$

$\phi({}^{s_+}_{b}) = {}^{s_+}_{b}$

$\phi({}^{0}_{p}) = {}^{0\ 0\ 2\ 1}_{p\ d\ b\ p} = {}^{0}_{p} D(2,0)$

$\phi({}^{s_+}_{p}) = {}^{0\ s_+\ 2\ s_+}_{p\ \ d\ \ q\ \ p} = P(s_+, 0) {}^{s_+}_{p}$

$\phi({}^{0}_{d}) = {}^{0\ 0\ 1\ 1}_{d\ b\ d\ q} = {}^{0}_{d} B(1,0)$

$\phi({}^{s_+}_{d}) = {}^{s_+}_{d}$

$\phi({}^{0}_{q}) = {}^{0\ 0\ 2\ 1}_{q\ b\ d\ q} = {}^{0}_{q} B(2,0)$

$\phi({}^{s_+}_{q}) = {}^{0\ s_+\ 2\ s_+}_{q\ \ b\ \ p\ \ q} = Q(s_+, 0) {}^{s_+}_{q}$

Lemma 8. $B(s_+, n) = B(s_+, 0)\phi(B(s_+, n-1))$
$D(s_+, n) = D(s_+, 0)\phi(D(s_+, n-1))$
$P(s_+, n) = \phi(P(s_+, n-1))P(s_+, 0)$
$Q(s_+, n) = \phi(Q(s_+, n-1))Q(s_+, 0)$

Proof. It is enough to prove it for $B(s_+, n)$ and $P(s_+, n)$, the other cases can be proved by symmetry.

$$\begin{aligned} B(s_+,0)\phi(B(s_+,n)) &= B(s_+,0)\phi(B(s_+,n-1) {}^{0}_{b} D(1,n-1) {}^{s_+}_{d} Q(1,n-1) {}^{1}_{q}) \\ &= B(s_+,0)\phi(B(s_+,n-1)) {}^{0}_{b} D(1,0)\phi(D(1,n-1)) {}^{s_+}_{d} \phi(Q(1,n-1))Q(1,0) {}^{1}_{q}) \\ &= B(s_+,n) {}^{0}_{b} D(1,n) {}^{s_+}_{d} Q(1,n) {}^{1}_{q} = B(s_+, ns_+1) \end{aligned}$$

$$\begin{aligned} \phi(P(s_+,n))P(s_+,0) &= \phi({}^{0}_{p} D(2,n-1) {}^{s_+}_{d} Q(2,n-1) {}^{2}_{q} P(s_+,n-1))P(s_+,0) \\ &= {}^{0}_{p} D(2,0)\phi(D(2,n-1)) {}^{s_+}_{d} \phi(Q(2,n-1))Q(2,0) {}^{2}_{q} \phi(P(s_+,n-1))P(s_+,0) \\ &= {}^{0}_{p} D(2,n) {}^{s_+}_{d} Q(2,n) {}^{2}_{q} P(s_+,n) = P(s_+, n+1) \end{aligned}$$

□

Theorem 3. *The t-shift of SMART is the closure of a fixed point of substitution ϕ.*

Proof. It is enough to prove that $\phi^n(\begin{smallmatrix}0\\b\end{smallmatrix}) = \begin{smallmatrix}0\\b\end{smallmatrix} D(1, n-1)$ for all $n \in \mathbb{N}$, because, from lemma 7, $\begin{smallmatrix}0\\b\end{smallmatrix} D(1, n-1)$ is the trace of $\left(\begin{smallmatrix}0 & 0^\omega\\b\end{smallmatrix}\right)$ over the first steps and, as the configuration is transitive, the orbit of this configuration is dense. We will prove it by induction.

Base of induction: $\phi(\begin{smallmatrix}0\\b\end{smallmatrix}) = \begin{smallmatrix}0&0&1&1\\b&d&b&p\end{smallmatrix} = \begin{smallmatrix}0\\b\end{smallmatrix} D(1, 0)$

Induction hypothesis: $\phi^n(\begin{smallmatrix}0\\b\end{smallmatrix}) = \begin{smallmatrix}0\\b\end{smallmatrix} D(1, n-1)$

Induction thesis:

$$\begin{aligned}\phi^{n+1}(\begin{smallmatrix}0\\b\end{smallmatrix}) &= \phi(\phi^n(\begin{smallmatrix}0\\b\end{smallmatrix})) \text{ // Induction hypothesis}\\ &= \phi(\begin{smallmatrix}0\\b\end{smallmatrix} D(1, n-1))\\ &= \begin{smallmatrix}0\\b\end{smallmatrix} D(1, 0)\phi(D(1, n-1)) \text{ // Lemma 8}\\ &= \begin{smallmatrix}0\\b\end{smallmatrix} D(1, n)\end{aligned} \tag{10}$$

□

3. An application of SMART

We will prove the second part of the conjecture in [8]: *"it is undecidable whether a given complete RTM admits a periodic configuration"*.
In order to ease the understanding of the proof, we will present two key proof techniques.

3.1. Proof techniques

3.1.1. Reversing the computation

This technique is used in [8]. It takes a reversible Turing machine $M = (Q, \Sigma, \delta)$, and creates a new reversible and complete machine $M' = (Q \times \{-, +\}, \Sigma, \delta')$, where $(r, +)$ and $(r, -)$ states represent M in state q running forwards or backwards in time, respectively. If at some iteration no instruction of M can be applied, the time direction is switched.
It is noteworthy to say that we can choose to switch the direction on just a set of pairs (state,symbol), defining in this way a possibly incomplete machine. For example, we can just switch the sign on the halting state of M; then M' will come back to the initial configuration if M halts from it.

3.1.2. Embedding

This technique consists in inserting a machine M inside a machine N to produce a new machine $\overline{N}$, in such a way that some property of M will be translated into a property of $\overline{N}$. In order to do this, some states of N are duplicated and connected to particular states of M.
For example, in order to insert $M = (Q, \Sigma, \delta)$ inside $N = (Q', \Sigma, \delta')$, we select a state $r \in Q'$ and split it into two new states, r' and r'', and connect the input instructions of r to r' and the outputs to r''. Now one can connect r' with the starting state of M, and r'' with the halting state of M. The connection can be done by a no movement and no writing instruction $((r', *, *, r_0, 0))$. We can also connect other states of M to those of N. In order to do this, we can use another state from N or we can split one of the new states r' or r'' into series.

3.2. Undecidability of the aperiodicity of complete reversible Turing machines

In [8], it is proved that the aperiodicity of (non-complete) reversible Turing machines is undecidable. There the proof is performed by reduction from the **reachability problem of aperiodic reversible Turing machines**: *Given an aperiodic and reversible Turing machine $M = (Q, \Sigma, \delta)$ and two states r_1, r_2, decide whether from r_1 M can reach r_2 or not*, which is proved undecidable in the same paper. The proof uses the technique of "reversing the computation" between r_1 and r_2 in order to define a reversible machine which has a periodic point if and only if (M, r_1, r_2) satisfies *reachability*. Now we will combine this idea with the

technique of "embedding" and the SMART machine to define a complete reversible machine with the same characteristic.

Definition 1. *A state $r' \in Q$ of a Turing machine $M = (Q, \Sigma, \delta)$ is said to be* defective *if there exists a symbol $s' \in \Sigma$ such that there exists no instruction (r, s, s', r', c), for no $r \in Q$, $s \in \Sigma$ and $c \in \{-1, 0, +1\}$. Analogously, we say that r' is an* error state *if there exists a symbol $s' \in \Sigma$ such that there exists no instruction (r', s', s, r, c), for no $r \in Q$, $s \in \Sigma$ and $c \in \{-1, 0, +1\}$.*

Theorem 4. *It is undecidable whether a given complete RTM admits a periodic orbit.*

Proof. Let us take an aperiodic RTM machine $M = (Q, \Sigma, \delta)$ and two states r_1, $r_2 \in Q$. Let us suppose that M has m defective states and n error states. Now, let us remove all transitions starting at state r_2 and all transitions arriving to state r_1. Using the technique of "reversing the computation", we create a new RTM machine $M' = (Q \times \{+, -\}, \Sigma, \delta')$ such that machine M is simulated forwards and backwards in time as we explained before, but now the direction is switched from - to + only in state r_1 and from + to - in state r_2. The machine M' is next duplicated to obtain two machines M_1' and M_2'. We will denote their forward and backward parts by $M_i'+$ and $M_i'-$ respectively.
The embedding into SMART is performed as follows. We invite the reader to look at figure 3 for a better understanding. We first split two states of SMART as many times as needed to have the necessary connections. We will denote by q_i', q_i'', with $i \in \{1, ..., m\}$ the set of split states of the first group, and by p_j' and p_j'', with $j \in \{1, ..., n\}$, the set of split states of the second group.

- States q_i' are connected to defective states of $M_1'+$ and states p_j' are connected to $M_1'-$.
- Error states of machine M_1' are directly connected to defective states of machine M_2' (going from $M_1'+$ to $M_2'-$, and from $M_1'-$ to $M_2'+$).
- Error states of machine $M_2'-$ are connected to states q_i'', and error states of $M_2'+$ are connected to p_j''.

The obtained machine will be called $\overline{SMART}$.
We first remark that if the machine M can reach r_2 from r_1, then $\overline{SMART}$ has a periodic point. Now let us suppose that r_2 is not reachable from r_1 through M.
This new machine is constructed such that, if we enter M_1' through one of the split states r_i' and we exit, then we exit by r_i'' and we find that the tape and the position of the head are not modified. We call this property *innocuity* of the embedding.
Let us prove innocuity for the split states q_i' (the proof for states p_j' is analogous). If we enter $M_1'+$ by state q_i' we have three possibilities:

- The first one is to stay inside $M_1'+$ and never exit.
- The second is to go to $M_1'-$ through state r_2; then the dynamics is reversed an we are forced to go to $M_2'+$. Now the dynamics is repeated and we go again to state r_2 of M_2' to finally exit by state q_i''. As any computation is next reversed, the tape and the position of the head do not change.
- The third is to go through error states of $M_1'+$ to $M_2'-$. Here the computation is reversed to exit by state q_i''. Again the tape and the position of the head do not change.

Thus, if we start on a state of SMART, the SMART dynamics is respected (except in the case we enter an infinite computation of M at some moment, but since M is aperiodic, this is not a problem). Therefore, since SMART is aperiodic, no periodic points appear. Let us study now the case when we start inside M. Three possibilities appear again.

- The first one is to exit the M_i' system. Then we arrive to the previous case, and we already know that the dynamics will not be periodic.

- The second is to fall into an infinite computation inside one of the parts of M'_1 or M'_2. This again cannot be periodic because M has no periodic points.
- The third is to move internally through the different machines M'_1+, M'_1-, M'_2+ and M'_2-. But, by construction, the only way of doing this is to alternate between M'_i+ and M'_i-, and this needs to go from r_1 to r_2, which is supposed impossible.

We conclude that r_2 is not reachable from r_1 by M, if and only if $\overline{SMART}$ has no periodic points. □

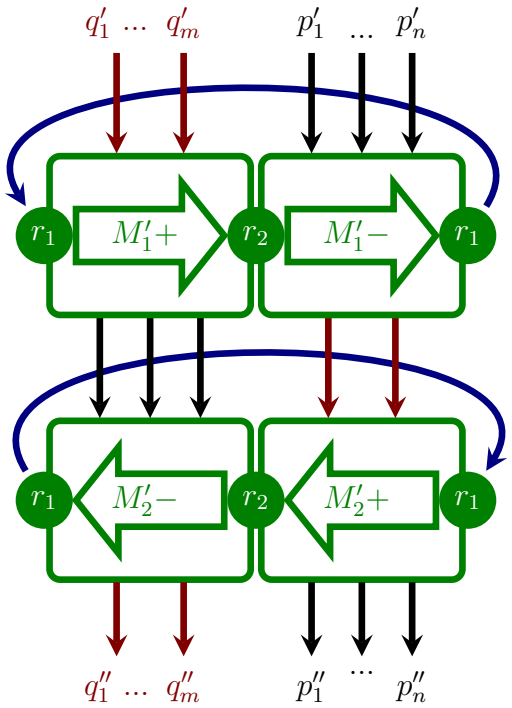

Figure 3: Embedding used in the proof of Theorem 4

[1] Birk, G.: Quelques thÃĽorÃĺmes sur les mouvements des systÃĺmes dynamiques. Bull. Soc. math. France **40** (1912) 305–323
[2] Blondel, V.D., Cassaigne, J., Nichitiu, C.: On the presence of periodic configurations in Turing machines and in counter machines. Theoret. Comput. Sci. **289** (2002) 573–590
[3] Oprocha, P.: On entropy and Turing machine with moving tape dynamical model. Nonlinearity **19** (October 2006) 2475–2487
[4] Jeandel, E.: Computability of the entropy of one-tape Turing machines. In: STACS. (2014)
[5] Gajardo, A., Mazoyer, J.: One head machines from a symbolic approach. Theor. Comput. Sci. **370** (2007) 34–47
[6] Gajardo, A., Guillon, P.: Zigzags in Turing machines. In Ablayev, F., Mayr, E., eds.: CSR. Volume 6072 of Lecture Notes in Computer Science., Springer (2010) 109–119
[7] Hooper, P.K.: The undecidability of the Turing machine immortality problem. Journal of Symbolic Logic **31**(2) (06 1966) 219–234
[8] Kari, J., Ollinger, N.: Periodicity and immortality in reversible computing. In Ochmanski, E., Tyszkiewicz, J., eds.: MFCS. Volume 5162 of Lecture Notes in Computer Science., Springer (2008) 419–430
[9] Kůrka, P.: On topological dynamics of Turing machines. Theoret. Comput. Sci. **174**(1-2) (1997) 203–216
[10] Gajardo, A., Kari, J., Moreira, A.: On time-symmetry in cellular automata. Journal of Computer and System Sciences (78) (2012) 1115–1126

Appendix A. Formal proof of combinatorial lemmas with the Coq proof assistant

The key ingredients of the paper are technical combinatorial lemmas about configurations accessibility. These lemma are handled by a careful case analysis and properly defined inductive properties. We started to investigate how such tedious lemmas can be completely and formally formalized and proved using the Co proof assistant software. The following pages provide the reviewer with complete proof of key lemmas of both the aperiodicity and the time-symmetry of the SMART machine. It is not intended to be published with the main content of the paper but rather provided as an hint to the reviewer that case analysis was indeed check up to the last detail.

The source code is organized as follows:

Tape is a formalization of infinite tapes of symbols (essentially streams) equipped with an adequate equivalence relation.

SMART defines the transition rule of the SMART machine.

Turing is a formalization of the machine dynamics: single and multi-step transitions eqquiped with an adequate equivalence relation on configurations.

SMARTaper states and proves the key lemmas for aperidocity: lemma PQBD corresponds to lemma 1, lemma srun corresponds to lemma 2, lemma irun corresponds to the main argument of lemma 3.

SMARTts states and proves time-symmetry of the SMART machine (lemma inv_h and theorem timesym).

Library Tape

```
Require Export Relation_Definitions.
Require Export Setoid.

Set Implicit Arguments.

Section DefEq.

Variable letter : Type.

CoInductive tape : Type := Cell { hd: letter; tl: tape }.

CoInductive Eqtape t t' : Prop :=
  eqtape : hd t = hd t' → Eqtape (tl t) (tl t') → Eqtape t t'.

Lemma Eqtape_refl : ∀ t, Eqtape t t.
cofix t; intros; constructor.
reflexivity.
destruct t0.
simpl.
apply t.
Qed.

Lemma Eqtape_sym : ∀ t t', Eqtape t t' → Eqtape t' t.
cofix t; intros; constructor.
destruct H.
rewrite H; reflexivity.
destruct t'.
destruct t0.
simpl.
destruct H.
simpl in H0.
apply t.
assumption.
Qed.

Lemma Eqtape_trans : ∀ t t' t'', Eqtape t t' → Eqtape t' t'' → Eqtape t t''.
cofix t; intros; constructor.
destruct H.
destruct H0.
rewrite H; rewrite H0; reflexivity.
destruct t0.
destruct t'.
destruct t''.
simpl.
destruct H.
destruct H0.
simpl in H1.
simpl in H2.
apply t with (t' := t'); assumption.
Qed.

End DefEq.

Add Parametric Relation A : (tape A) (@Eqtape A)
  reflexivity proved by (@Eqtape_refl A)
  symmetry proved by (@Eqtape_sym A)
  transitivity proved by (@Eqtape_trans A)
  as eqtape_equiv.
```

```
Add Parametric Morphism A : (@tl A)
    with signature (@Eqtape A) ==> (@Eqtape A)
    as tl_morph.
intros.
destruct H.
assumption.
Qed.

Notation "x :: xs" := (Cell x xs).
Notation "x == y" := (Eqtape x y) (at level 70).

Section Map.
Variables A B : Type.
Variable f : A → B.
CoFixpoint map (s : tape A) : tape B := f (hd s)::map (tl s).
End Map.

Add Parametric Morphism A B f : (@map A B f)
    with signature (@Eqtape A) ==> (@Eqtape B)
    as map_morph.
cofix x; intros; constructor.
destruct x0.
destruct y.
simpl.
destruct H.
simpl in H.
rewrite H.
reflexivity.
destruct x0.
destruct y.
simpl.
apply x.
destruct H.
simpl in H0.
assumption.
Qed.

Definition compose {A B C} (f : B → C) (g : A → B) := fun x ⇒ f (g x).

Lemma map_comp : ∀ A B C (f : B → C) (g : A → B) t, map f (map g t) == map (compose f g) t.
intros A B C f g.
cofix t; intros; constructor.
reflexivity.
simpl.
apply t.
Qed.

Unset Implicit Arguments.
```

Library SMART

```
Require Export Tape.

Inductive letter : Set := L0 | L1 | L2.
Inductive state : Set := Qp | Qq | Qb | Qd.
Inductive direction : Set := Left | Right.

Definition deltaQ x := match x with
  | (Qb, L0) ⇒ Qd
  | (Qb, _) ⇒ Qp
  | (Qd, L0) ⇒ Qb
  | (Qd, _) ⇒ Qq
  | (Qq, L2) ⇒ Qp
  | (Qq, _) ⇒ Qb
  | (Qp, L2) ⇒ Qq
  | (Qp, _) ⇒ Qd
  end.

Definition deltaL x := match x with
  | (Qb, L0) ⇒ L1
  | (Qb, l) ⇒ l
  | (Qd, L0) ⇒ L1
  | (Qd, l) ⇒ l
  | (Qq, L2) ⇒ L0
  | (Qq, L0) ⇒ L2
  | (Qq, L1) ⇒ L0
  | (Qp, L2) ⇒ L0
  | (Qp, L0) ⇒ L2
  | (Qp, L1) ⇒ L0
  end.

Definition deltaD x := match x with
  | (Qb, _) ⇒ Right
  | (Qd, _) ⇒ Left
  | (Qq, L2) ⇒ Right
  | (Qq, _) ⇒ Left
  | (Qp, L2) ⇒ Left
  | (Qp, _) ⇒ Right
  end.
```

```
Library Turing
Require Export SMART.

Inductive cfg : Set := Cfg { ltape: tape letter; st: state; rtape: tape letter }.

Definition goLeft c := Cfg (tl (ltape c)) (st c) ((hd (ltape c)) :: (rtape c)).

Definition goRight c := Cfg ((hd (rtape c)) :: (ltape c)) (st c) (tl (rtape c)).

Definition point c := (st c, hd (rtape c)).

Definition step c :=
  let p := point c in
  match deltaD p with
    | Left ⇒
        goLeft (Cfg (ltape c) (deltaQ p) ((deltaL p) :: (tl (rtape c))))
    | Right ⇒
        goRight (Cfg (ltape c) (deltaQ p) ((deltaL p) :: (tl (rtape c))))
  end.

Fixpoint nstep n c :=
  match n with
  | 0 ⇒ c
  | S n ⇒ nstep n (step c)
  end.

Lemma nstep_decomp : ∀ m n c, nstep n (nstep m c) = nstep (n+m) c.
Proof.
induction m.
- intros.
  simpl.
  replace (n + 0) with n by auto.
  reflexivity.
- intros.
  rewrite ← plus_n_Sm.
  simpl.
  apply IHm.
Qed.

Inductive samecfg : cfg → cfg → Prop :=
  samecfg0 : ∀ ta ta' tb tb' s, ta == tb → ta' == tb' → samecfg (Cfg ta s ta') (Cfg tb s tb').

Lemma samecfg_refl : ∀ t, samecfg t t.
Proof.
intros [t s t'].
apply samecfg0; reflexivity.
Qed.

Lemma samecfg_sym : ∀ t t', samecfg t t' → samecfg t' t.
Proof.
intros [ta sa ta'] [tb sb tb'] H.
destruct H.
apply samecfg0.
- rewrite H; reflexivity.
- rewrite H0; reflexivity.
Qed.

Lemma samecfg_trans : ∀ t t' t'', samecfg t t' → samecfg t' t'' → samecfg t t''.
Proof.
```

```
intros [ta sa ta'] [tb sb tb'] [tc sc tc'] H H0.
inversion H.
inversion H0.
apply samecfg0.
- rewrite H3; rewrite H11; reflexivity.
- rewrite H8; rewrite H16; reflexivity.
Qed.

Add Relation cfg samecfg
  reflexivity proved by samecfg_refl
  symmetry proved by samecfg_sym
  transitivity proved by samecfg_trans
  as samecfg_equiv.

Notation "x === y" := (samecfg x y) (at level 70).

Add Morphism step
    with signature samecfg ==> samecfg
    as step_morph.
Proof.
intros [[a ta] sa [a' ta']] [[b tb] sb [b' tb']] H.
inversion H.
inversion H2.
inversion H7.
simpl in H8.
simpl in H9.
simpl in H10.
simpl in H11.
rewrite H8.
rewrite H10.
induction sb; induction b'; unfold step; unfold goRight; unfold goLeft; simpl;
(apply samecfg0; (assumption —— (constructor; simpl; [reflexivity | constructor; simpl; [ reflexivity
| assumption ]]))).
Qed.

Add Parametric Morphism n : (nstep n)
    with signature samecfg ==> samecfg
    as nstep_morph.
Proof.
induction n; intros.
- simpl; assumption.
- simpl.
  apply IHn.
  apply step_morph.
  assumption.
Qed.
```

```
Library SMARTaper
Require Export Turing.

Fixpoint u n :=
  match n with
  | 0 ⇒ 3
  | S n ⇒ 3+3*(u n)
  end.

Lemma uSn : ∀ n, u (S n) = 3 + 3*(u n).
Proof.
auto.
Qed.

Require Import Arith.

Lemma uSno : ∀ n, u (S n) = u n + 1 + u n + 1 + u n + 1.
Proof.
intro.
rewrite uSn.
unfold mult.
ring.
Qed.

Lemma uSnu : ∀ n, u (S n) = 1 + u n + 1 + u n + 1 + u n.
Proof.
intro.
rewrite uSn.
unfold mult.
ring.
Qed.

Fixpoint zzz t n :=
  match n with
  | 0 ⇒ t
  | S n ⇒ L0::(zzz t n)
  end.

Lemma zzzL0 : ∀ t n, zzz (L0 :: t) n = L0 :: zzz t n.
Proof.
induction n.
 - auto.
 - simpl; rewrite IHn; reflexivity.
Qed.

Definition conf_p n x t t' := Cfg t Qp (zzz (x::t') (n+1)).
Definition img_p n x t t' := Cfg (zzz t (n+1)) Qp (x::t').

Definition conf_q n x t t' := Cfg (zzz (x::t) n) Qq (L0::t').
Definition img_q n x t t' := Cfg t Qq (x::(zzz t' (n+1))).

Definition conf_b n x y t t' := Cfg (zzz (x::t) n) Qb (L0::y::t').
Definition img_b n x y t t' := Cfg t Qb (x::(zzz (y::t') (n+1))).

Definition conf_d n x y t t' := Cfg (x::t) Qd (zzz (y::t') (n+1)).
Definition img_d n x y t t' := Cfg (zzz (x::t) (n+1)) Qd (y::t').

Definition P n := ∀ x t t', ((x = L1) ∨ (x = L2)) → nstep (u n) (conf_p n x t t') = img_p n x t t'.
Definition Q n := ∀ x t t', ((x = L1) ∨ (x = L2)) → nstep (u n) (conf_q n x t t') = img_q n x t t'.
Definition B n := ∀ x y t t', ((y = L1) ∨ (y = L2)) → nstep (u n) (conf_b n x y t t') = img_b n x y t t'.
```

```
Definition D n := ∀ x y t t', ((x = L1) ∨ (x = L2)) → nstep (u n) (conf_d n x y t t') = img_d n x y t t'.
Lemma stepp : ∀ n x t t', (step (conf_p (S n) x t t')) = conf_d n L2 x t t'.
Proof.
auto.
Qed.
Lemma stepq : ∀ n x t t', (step (conf_q (S n) x t t')) = conf_b n x L2 t t'.
Proof.
auto.
Qed.
Lemma stepid : ∀ n x y t t', (y = L1 ∨ y = L2) → (step (img_d n x y t t')) = conf_q n x t (y::t').
Proof.
intros n x y t t' [H | H]; rewrite H;
unfold img_d, conf_q, step, goLeft, goRight; simpl;
replace (n + 1) with (S n) by ring;
auto.
Qed.
Lemma stepib : ∀ n x y t t', (x = L1 ∨ x = L2) → (step (img_b n x y t t')) = conf_p n y (x::t) t'.
Proof.
intros n x y t t' [H | H]; rewrite H;
unfold img_b, conf_p, step, goLeft, goRight; simpl;
replace (n + 1) with (S n) by ring;
auto.
Qed.
Lemma stepiq : ∀ n x t t', step (img_q n L2 t (x::t')) = conf_p n x (L0::t) t'.
auto.
Qed.
Lemma stepiq' : ∀ n x y t t', step (img_q n L1 (x :: t) (y :: t')) = img_b (S n) x y t t'.
auto.
Qed.
Lemma stepip : ∀ n x t t', step (img_p n L2 (x::t) t') = conf_q n x t (L0::t').
Proof.
intros n x t t';
unfold img_p, conf_q, step, goLeft, goRight; simpl;
replace (n + 1) with (S n) by ring;
auto.
Qed.
Lemma stepib' : ∀ n x y t t', step (img_b n L0 y (x :: t) t') = conf_d n L1 y (x::t) t'.
Proof.
auto.
Qed.
Lemma stepid' : ∀ n x t t', step (img_d n x L0 t t') = conf_b n x L1 t t'.
Proof.
intros n x t t';
unfold img_d, conf_b, step, goLeft, goRight; simpl;
replace (n + 1) with (S n) by ring;
simpl; reflexivity.
Qed.
Lemma stepip' : ∀ n x y t t', step (img_p n L1 (x :: t) (y :: t')) = img_d (S n) x y t t'.
Proof.
```

```
auto.
Qed.
Lemma confbSn : ∀ n x y t t', conf_b (S n) x y t t' = conf_b n L0 y (x::t) t'.
Proof.
intros; unfold conf_b; simpl; rewrite zzzL0; reflexivity.
Qed.
Lemma confdSn : ∀ n x y t t', conf_d (S n) x y t t' = conf_d n x L0 t (y::t').
Proof.
intros; unfold conf_d; simpl; rewrite zzzL0; reflexivity.
Qed.
Lemma PQBD : ∀ n, P n ∧ Q n ∧ B n ∧ D n.
Proof.
induction n.
 - split; [—split; [—split]]; unfold P, Q, B, D; intros; destruct H; rewrite H; compute; reflexivity.
 - destruct IHn as [ Pn [ Qn [ Bn Dn ] ] ]; split; [—split; [—split]].
   + unfold P;
     intros x t t' [ H | H ]; rewrite H;
     rewrite uSno;
     do 5 rewrite ← nstep_decomp;
     simpl;
     rewrite stepp;
     unfold D in Dn; rewrite Dn by auto;
     rewrite stepid by auto;
     unfold Q in Qn; rewrite Qn by auto;
     rewrite stepiq;
     unfold P in Pn; rewrite Pn by auto;
     unfold img_p;
     simpl;
     rewrite zzzL0;
     reflexivity.
   + unfold Q;
     intros x t t' [ H | H ]; rewrite H;
     rewrite uSno;
     do 5 rewrite ← nstep_decomp;
     simpl;
     rewrite stepq;
     unfold B in Bn; rewrite Bn by auto;
     rewrite stepib by auto;
     unfold P in Pn; rewrite Pn by auto;
     rewrite stepip;
     unfold Q in Qn; rewrite Qn by auto;
     unfold img_q;
     simpl;
     rewrite zzzL0;
     reflexivity.
   + unfold B;
     intros x y t t' [ H | H ]; rewrite H;
     rewrite uSnu;
     do 5 rewrite ← nstep_decomp;
     simpl;
```

```
    rewrite confbSn;
    unfold B in Bn; rewrite Bn by auto;
    rewrite stepib';
    unfold D in Dn; rewrite Dn by auto;
    rewrite stepid by auto;
    unfold Q in Qn; rewrite Qn by auto;
    rewrite stepiq';
    reflexivity.
  + unfold D;
    intros x y t t' [ H | H ]; rewrite H;
    rewrite uSnu;
    do 5 rewrite ← nstep_decomp;
    simpl;
    rewrite confdSn;
    unfold D in Dn; rewrite Dn by auto;
    rewrite stepid';
    unfold B in Bn; rewrite Bn by auto;
    rewrite stepib by auto;
    unfold P in Pn; rewrite Pn by auto;
    rewrite stepip';
    reflexivity.
Qed.

Fixpoint v n :=
  match n with
  | 0 ⇒ 0
  | S n ⇒ v n + 2 × (u n) + 3
  end.

Lemma vSno : ∀ n, v (S n) = v n + 1 + u n + 1 + u n + 1.
Proof.
intro n; unfold v; ring.
Qed.

Definition img_p' n x t t' := Cfg (zzz t n) Qp (L0::x::t').
Definition img_q' n x t t' := Cfg (x::t) Qq (zzz t' (n+1)).

Definition P' n := ∀ x t t', ((x = L1) ∨ (x = L2)) → nstep (v n) (conf_p n x t t') = img_p' n x t t'.
Definition Q' n := ∀ x t t', ((x = L1) ∨ (x = L2)) → nstep (v n) (conf_q n x t t') = img_q' n x t t'.

Lemma IP : ∀ n, P n.
Proof.
intro n; apply PQBD.
Qed.

Lemma IQ : ∀ n, Q n.
Proof.
intro n; apply PQBD.
Qed.

Lemma IB : ∀ n, B n.
Proof.
intro n; apply PQBD.
Qed.

Lemma ID : ∀ n, D n.
Proof.
```

```
intro n; apply PQBD.
Qed.

Lemma lP' : ∀ n, P' n.
Proof.
induction n; unfold P'; intros x t t' H.
- compute; reflexivity.
- unfold P'.
  rewrite vSno;
  do 5 rewrite ← nstep_decomp;
  simpl;
  rewrite stepp;
  rewrite lD by auto;
  rewrite stepid by auto;
  rewrite lQ by auto;
  rewrite stepiq;
  unfold P' in IHn; rewrite IHn by auto;
  unfold img_p';
  simpl;
  rewrite zzzL0;
  reflexivity.
Qed.

Lemma lQ' : ∀ n, Q' n.
Proof.
induction n; unfold Q'; intros x t t' H.
- compute; reflexivity.
- unfold P';
  rewrite vSno;
  do 5 rewrite ← nstep_decomp;
  simpl;
  rewrite stepq;
  rewrite lB by auto;
  rewrite stepib by auto;
  rewrite lP by auto;
  rewrite stepip;
  unfold Q' in IHn; rewrite IHn by auto;
  unfold img_q';
  simpl;
  rewrite zzzL0;
  reflexivity.
Qed.

Definition w n := 1 + (u (S n)) + 1 + 1 + (u n).

Definition conf_gp n x t t' := Cfg (x::t) Qp (zzz (L1::L0::t') (n+1)).
Definition img_gp n x t t' := Cfg (x::t) Qp (zzz (L1::t') (n+2)).

Lemma conf_gp_is_p : ∀ n x t t', conf_gp n x t t' = conf_p n L1 (x::t) (L0::t').
Proof.
intros; unfold conf_gp, conf_p; simpl.
reflexivity.
Qed.

Lemma img_gp_is_p : ∀ n x t t', img_gp n x t t' = conf_p (S n) L1 (x :: t) t'.
Proof.
```

```
intros; unfold img_gp, conf_p; simpl; replace (n + 2) with (S (n + 1)) by ring; simpl; reflexivity.
Qed.
Definition growP n := ∀ x t t', ((x = L1) ∨ (x = L2)) → nstep (w n) (conf_gp n x t t') = img_gp n x t t'.
Lemma gP : ∀ n, growP n.
Proof.
unfold growP; intros n x t t' H.
unfold w;
do 4 rewrite ← nstep_decomp.
rewrite conf_gp_is_p.
rewrite IP by auto.
rewrite stepip' by auto.
rewrite stepid'.
rewrite IB by auto.
rewrite stepib by auto.
symmetry; apply img_gp_is_p.
Qed.
CoFixpoint blanks := L0 :: blanks.
Lemma blanksdef : blanks == L0 :: blanks.
Proof.
constructor; simpl; reflexivity.
Qed.
Lemma samezzz : ∀ n t t', t == t' → zzz t n == zzz t' n.
Proof.
induction n; intros; simpl.
 - assumption.
 - constructor; simpl.
   + reflexivity.
   + apply IHn; assumption.
Qed.
Lemma run : ∀ n x, x = L1 ∨ x = L2 → nstep (w n) (conf_gp n x blanks blanks) === conf_gp (S n) x blanks
blanks.
Proof.
intros n x H.
rewrite gP by auto.
unfold img_gp, conf_gp; replace (n + 2) with (S (n + 1)) by ring; simpl.
constructor.
 - reflexivity.
 - constructor; simpl.
   + reflexivity.
   + apply samezzz.
     constructor; simpl.
     × reflexivity.
     × apply blanksdef.
Qed.
Fixpoint sw n := match n with
  | 0 ⇒ 0
  | S n ⇒ (w n) + (sw n)
end.
Lemma srun : ∀ n x, x = L1 ∨ x = L2 → nstep (sw n) (conf_gp 0 x blanks blanks) === conf_gp n x blanks
blanks.
```

```
Proof.
  intros n x H; induction n.
   - simpl; reflexivity.
   - replace (sw (S n)) with ((w n) + (sw n)) by auto.
     rewrite ← nstep_decomp.
     rewrite IHn by auto.
     apply run.
     assumption.
Qed.

Lemma start : nstep 9 (Cfg blanks Qp blanks) === conf_gp 0 L2 blanks blanks.
Proof.
compute.
constructor.
 - reflexivity.
 - constructor.
   + reflexivity.
   + simpl; constructor.
     × reflexivity.
     × simpl; apply blanksdef.
Qed.

Lemma irun : ∀ n, nstep (sw n + 9) (Cfg blanks Qp blanks) === conf_gp n L2 blanks blanks.
Proof.
intro n.
rewrite ← nstep_decomp.
rewrite start.
rewrite srun by auto.
reflexivity.
Qed.
```

Library SMARTsim

```
Require Import Turing.

Definition simq s := match s with
  | Qp ⇒ Qq
  | Qq ⇒ Qp
  | Qd ⇒ Qb
  | Qb ⇒ Qd
end.

Lemma sqsq : ∀ s, simq (simq s) = s.
Proof.
intro s; case s; auto.
Qed.

Definition simc c := Cfg (tl (rtape c)) (simq (st c)) ((hd (rtape c)) :: (ltape c)).

Lemma simsim : ∀ c, simc (simc c) = c.
Proof.
unfold simc; intros [t s [a t']]; simpl; rewrite sqsq; reflexivity.
Qed.

Lemma simstep : ∀ c, step (simc c) = simc (step c).
Proof.
unfold step, simc, simq, goLeft, goRight; intros [[a t] s [b [c t']]]; case s; case b; simpl; reflexivity.
Qed.

Lemma simnstep : ∀ n c, nstep n (simc c) = simc (nstep n c).
Proof.
induction n; intro c; simpl.
 - reflexivity.
 - rewrite simstep; apply IHn.
Qed.
```

Library SMARTts

```
Require Import Turing.

Definition hQ s :=
  match s with
    | Qd ⇒ Qq
    | Qq ⇒ Qd
    | Qp ⇒ Qb
    | Qb ⇒ Qp
  end.

Lemma inv_hQ : ∀ s, hQ (hQ s) = s.
Proof.
intro s; case s; auto.
Qed.

Definition hL l :=
  match l with
    | L0 ⇒ L0
    | L1 ⇒ L2
    | L2 ⇒ L1
  end.

Lemma inv_hL : ∀ l, hL (hL l) = l.
Proof.
intro l; case l; auto.
Qed.

Definition thL t := map hL t.

Lemma inv_thL : ∀ t, (thL (thL t)) == t.
Proof.
cofix t; intros; constructor.
- simpl; rewrite inv_hL; reflexivity.
- destruct t0; apply t.
Qed.

Definition predir s :=
  match s with
    | Qp ⇒ Right
    | Qq ⇒ Left
    | Qb ⇒ Left
    | Qd ⇒ Right
  end.

Definition h c :=
  match predir (st c) with
    | Left ⇒ goRight (Cfg (thL (ltape c)) (hQ (st c)) (thL (rtape c)))
    | Right ⇒ goLeft (Cfg (thL (ltape c)) (hQ (st c)) (thL (rtape c)))
  end.

Lemma inv_h : ∀ c, h (h c) === c.
Proof.
induction c as [ [a t] s [b t'] ].
induction s; unfold h, goLeft, goRight, thL; simpl; rewrite inv_hL; replace (cofix map s := hL (hd s)
:: map (tl s)) with (map hL) by auto; constructor; (apply inv_thL —— (constructor; [ auto | apply
inv_thL])).
Qed.
```

```
Theorem timesym : ∀ c, h (step (h (step c))) === c.
Proof.
intro c.
induction c as [ [a [b t]] s [c [d t']] ].
induction s; induction a; induction b; induction c; induction d; (compute; replace (cofix map s :=
            match match s with
                  | l :: _ ⇒ l
                  end with
            | L0 ⇒ L0
            | L1 ⇒ L2
            | L2 ⇒ L1
            end :: map match s with
                        | _ :: t'0 ⇒ t'0
                        end) with (map hL) by auto; constructor; (constructor; [ reflexivity |
simpl; apply inv_thL ])).
Qed.

Definition pets c := h (step (h c)).

Lemma inv_step : ∀ c, pets (step c) === c.
unfold pets.
apply timesym.
Qed.
```